



File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

Save + Cut Copy Paste Undo Redo Run Stop Restart Markdown

```
In [1]: # importing libraries
import pandas as pd
import numpy as np           # For mathematical calculations
import seaborn as sns       # For data visualization
import matplotlib.pyplot as plt
import seaborn as sn        # For plotting graphs
%matplotlib inline
import warnings              # To ignore any warnings
warnings.filterwarnings("ignore")
```

```
In [2]: # Loading the data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

Let's check the features present in our data and then we will look at their data types.

```
In [3]: train.columns
```

```
Out[3]: Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
              'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
              'pdays', 'previous', 'poutcome', 'subscribed'],
              dtype='object')
```



```
In [4]: test.columns
```

```
Out[4]: Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
              'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
              'pdays', 'previous', 'poutcome'],
              dtype='object')
```

It can be inferred that **subscribed** is the target variable as it is not present in the test dataset. Let's look at the shape of the dataset.

```
In [5]: train.shape, test.shape
```

```
Out[5]: ((31647, 18), (13564, 17))
```

We have 17 independent variables and 1 target variable, i.e. subscribed in the train dataset. We have similar features in the test dataset as the train dataset except the subscribed. We will predict the subscribed with the help of model built using the train data.

Next, let's look at how many categorical and numerical variables are there in our dataset. We will look at their data types.



```
In [7]: #printing first five rows of the dataset
train.head()
```

Out[7]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	0	unknown
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	0	unknown
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	0	unknown
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	3	success
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	0	unknown

Univariate Analysis

Now Let's look at the distribution of our target variable, i.e. subscribed. As it is a categorical variable, let us look at its frequency table, percentage distribution and bar plot.

```
In [8]: train['subscribed'].value_counts()
```

```
Out[8]: no      27932
yes       3715
Name: subscribed, dtype: int64
```

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

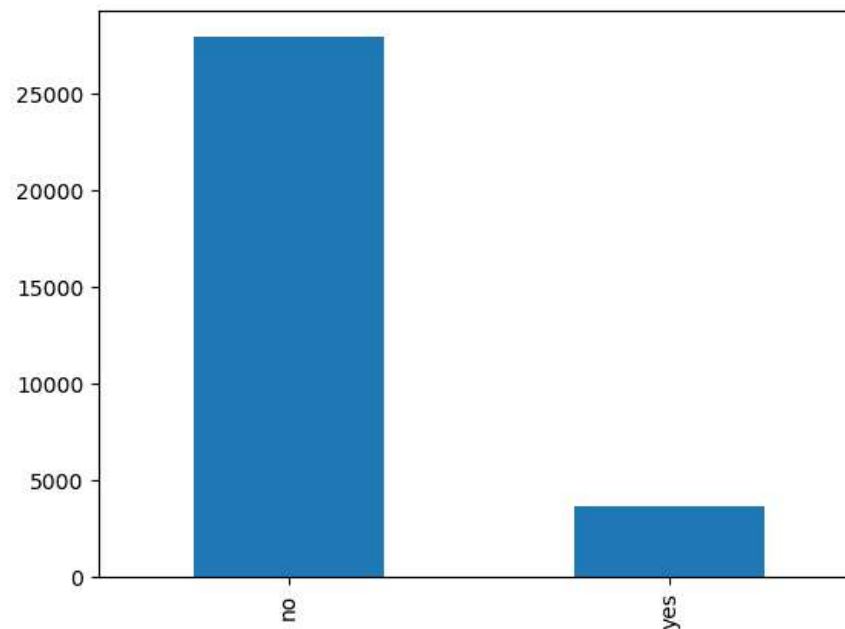
Run Code

```
In [9]: # Normalize can be set to True to print proportions instead of number
train['subscribed'].value_counts(normalize=True)
```

```
Out[9]: no    0.882611
        yes    0.117389
        Name: subscribed, dtype: float64
```

```
In [10]: # plotting the bar plot of frequencies
train['subscribed'].value_counts().plot.bar()
```

```
Out[10]: <Axes: >
```



So, 3715 users out of total 31647 have subscribed which is around 12%. Let's now explore the variables to have a better understanding of the dataset. We will first explore the variables individually using univariate analysis, then we will look at the relation between various independent variables and the target variable. We will also look at the correlation plot to see which variables affects the target variable most.

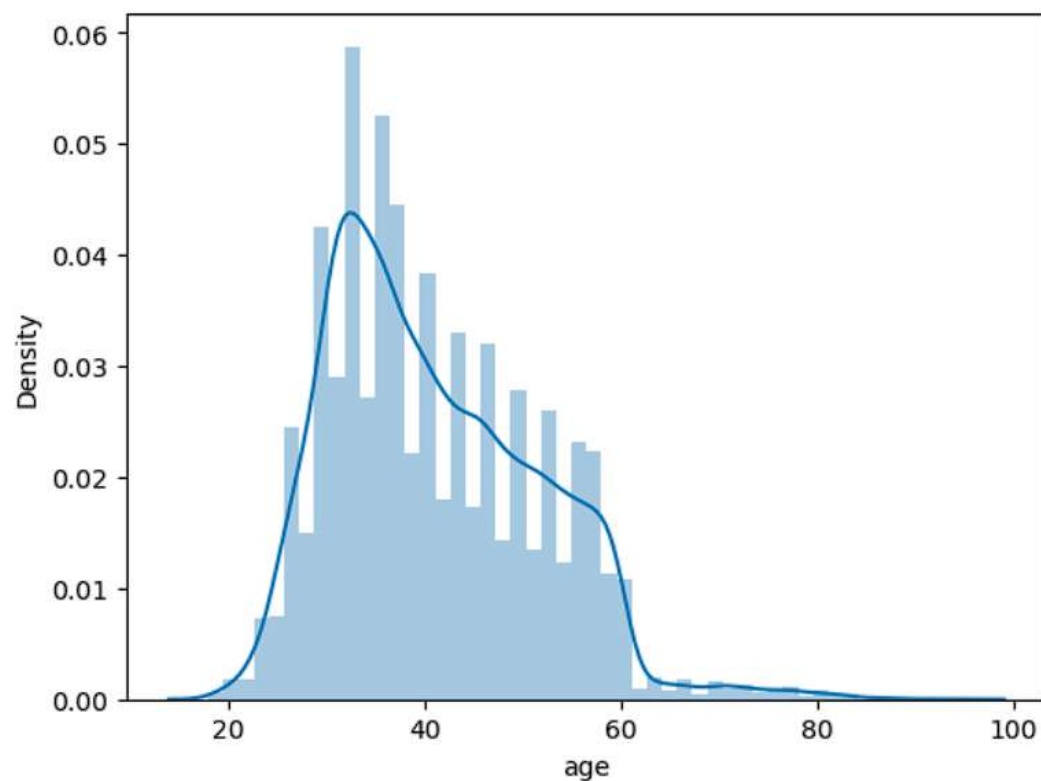
Let's first look at the distribution of age variable to see how many people belongs to a particular age group.

```
In [11]: sn.distplot(train["age"])
```

```
Out[11]: <Axes: xlabel='age', ylabel='Density'>
```

```
In [11]: ▶ sn.distplot(train["age"])
```

```
Out[11]: <Axes: xlabel='age', ylabel='Density'>
```

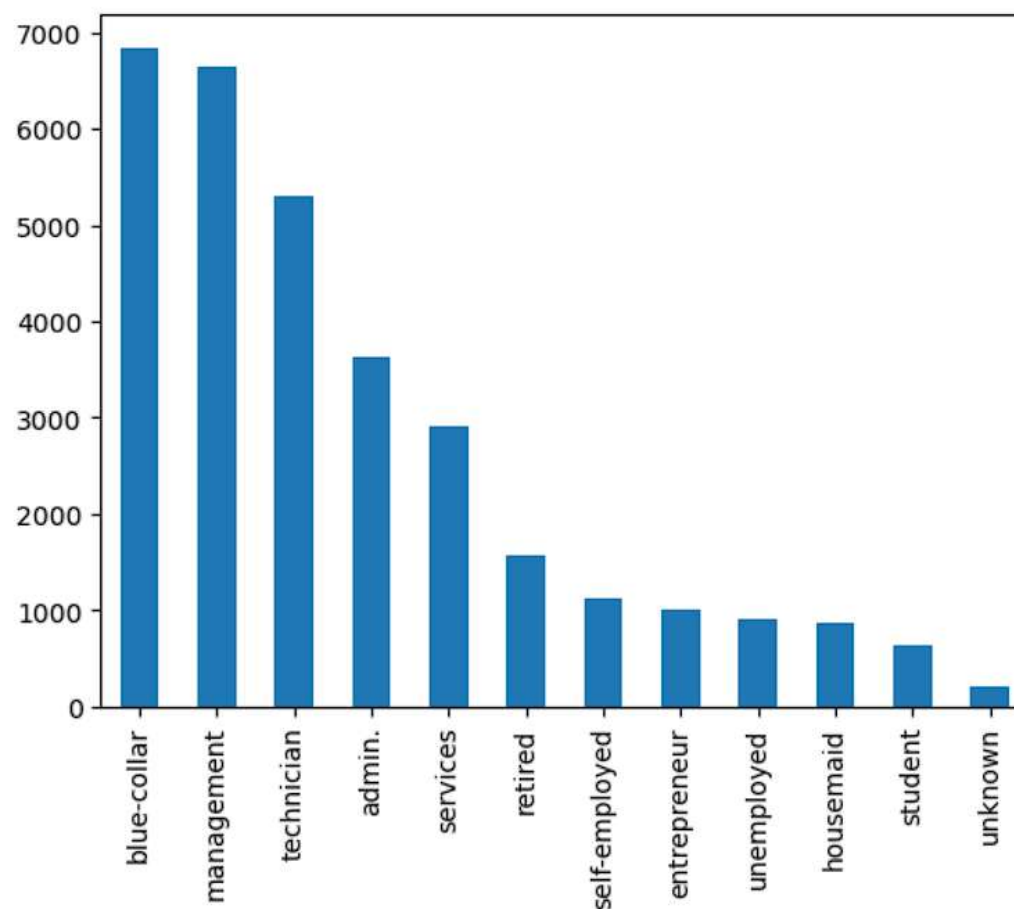


We can infer that most of the clients fall in the age group between 20-60. Now let's look at what are the different types of jobs of the clients. As job is a categorical variable, we will look at its frequency table

categorical variable, we will look at its frequency table

```
In [12]: train['job'].value_counts().plot.bar()
```

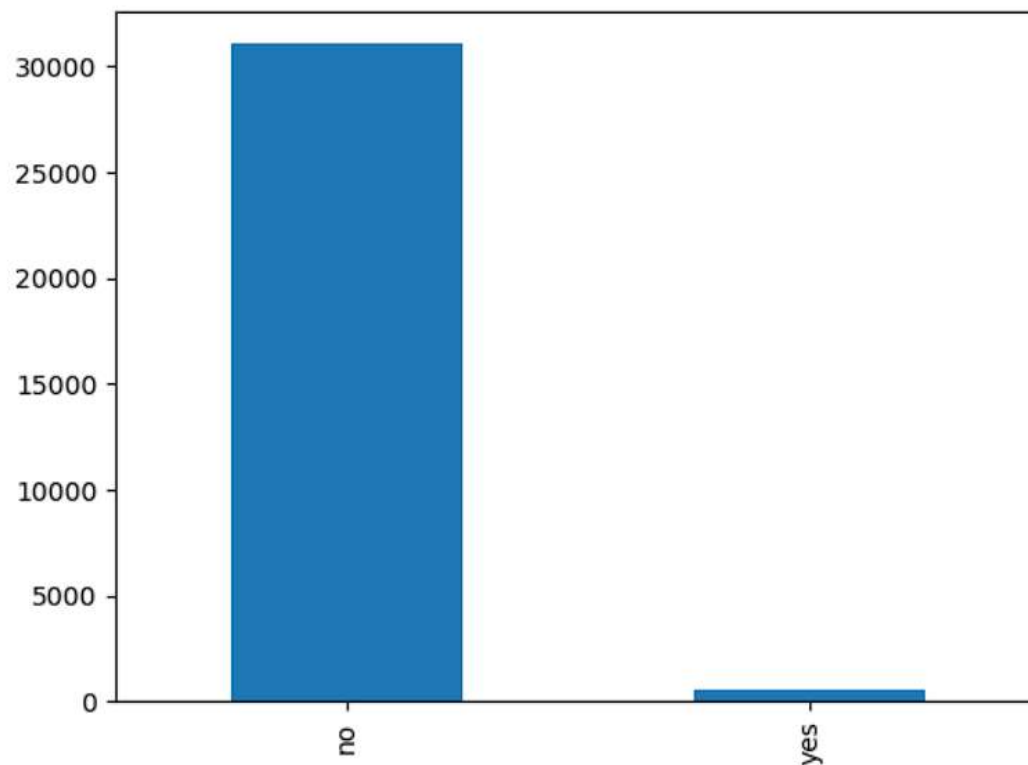
Out[12]: <Axes: >





```
In [13]: train['default'].value_counts().plot.bar()
```

Out[13]: <Axes: >



More than 90% of the clients have no default history. Now we will explore these variables against the target variable using bivariate analysis. We will make use of scatter plots for continuous or numeric variables and crosstabs for the categorical variables. Let's start with job and subscribed variable.



More than 90% of the clients have no default history. Now we will explore these variables against the target variable using bivariate analysis. We will make use of scatter plots for continuous or numeric variables and crosstabs for the categorical variables. Let's start with job and subscribed variable.

Bivariate Analysis

```
In [14]: print(pd.crosstab(train['job'],train['subscribed']))

job=pd.crosstab(train['job'],train['subscribed'])
job.div(job.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))
plt.xlabel('Job')
plt.ylabel('Percentage')
```

subscribed	no	yes
job		
admin.	3179	452
blue-collar	6353	489
entrepreneur	923	85
housemaid	795	79
management	5716	923
retired	1212	362
self-employed	983	140
services	2649	254
student	453	182
technician	4713	594
unemployed	776	129
unknown	180	26

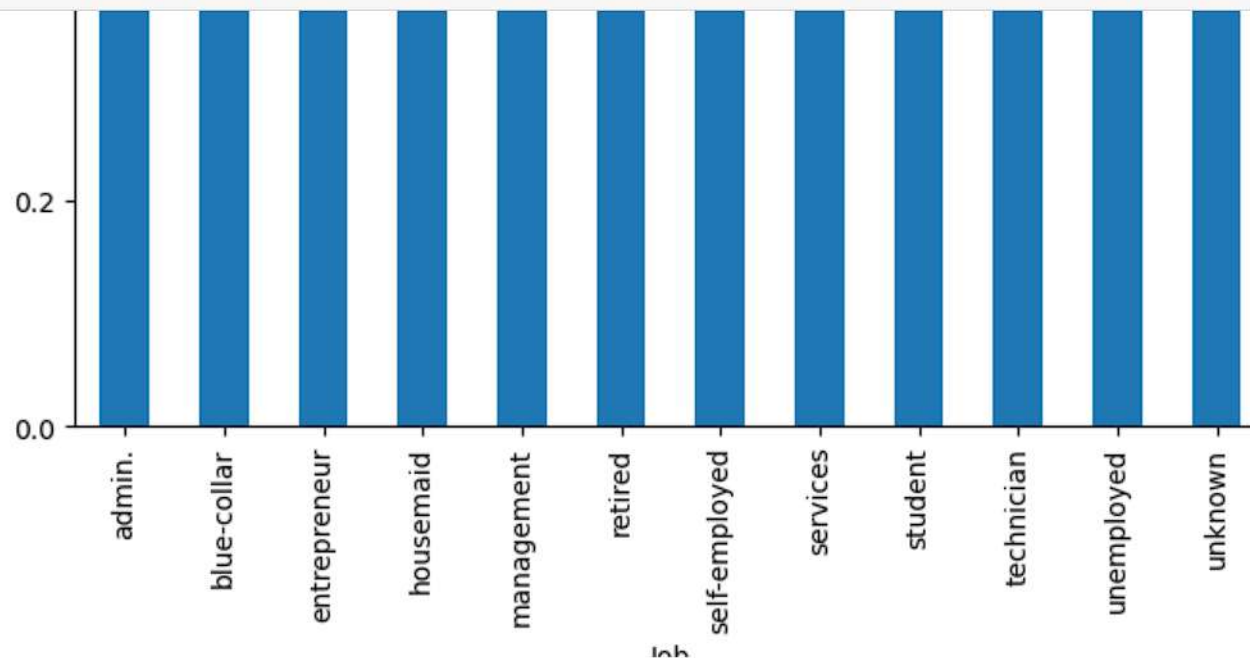
Out[14]: Text(0, 0.5, 'Percentage')

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel) ○

```
job.div(job.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))  
plt.xlabel('Job')  
plt.ylabel('Percentage')
```



From the above graph we can infer that students and retired people have higher chances of subscribing to a term deposit, which is surprising as students generally do not subscribe to a term deposit. The possible reason is that the number of students in the dataset is less and comparatively to other job types, more students have subscribed to a term deposit.

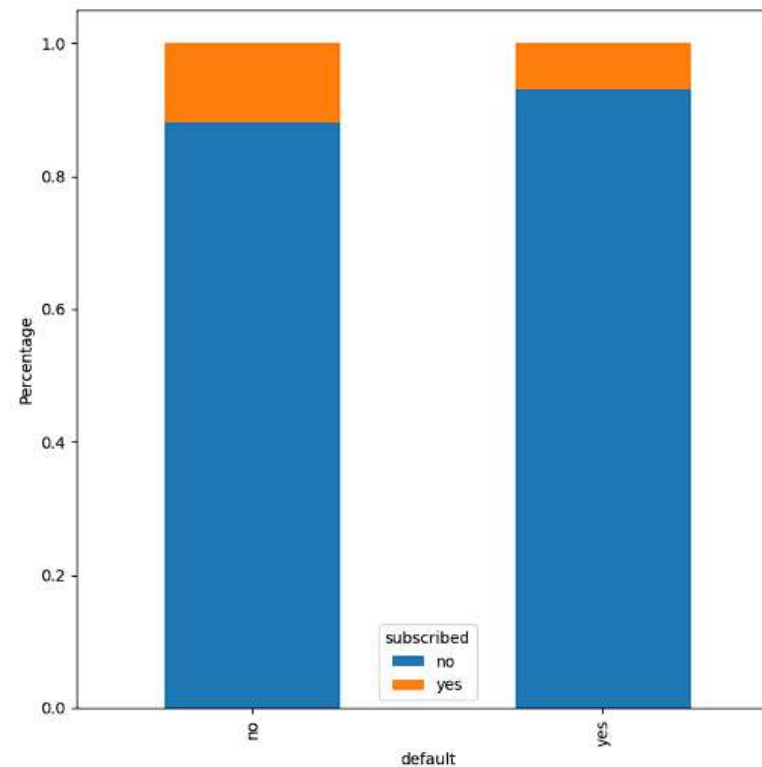
Next, let's explore the default variable against the subscribed variable.

```
In [15]: print(pd.crosstab(train['default'],train['subscribed']))

default=pd.crosstab(train['default'],train['subscribed'])
default.div(default.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))
plt.xlabel('default')
plt.ylabel('Percentage')
```

```
subscribed    no    yes
default
no           27388  3674
yes           544    41
```

Out[15]: Text(0, 0.5, 'Percentage')



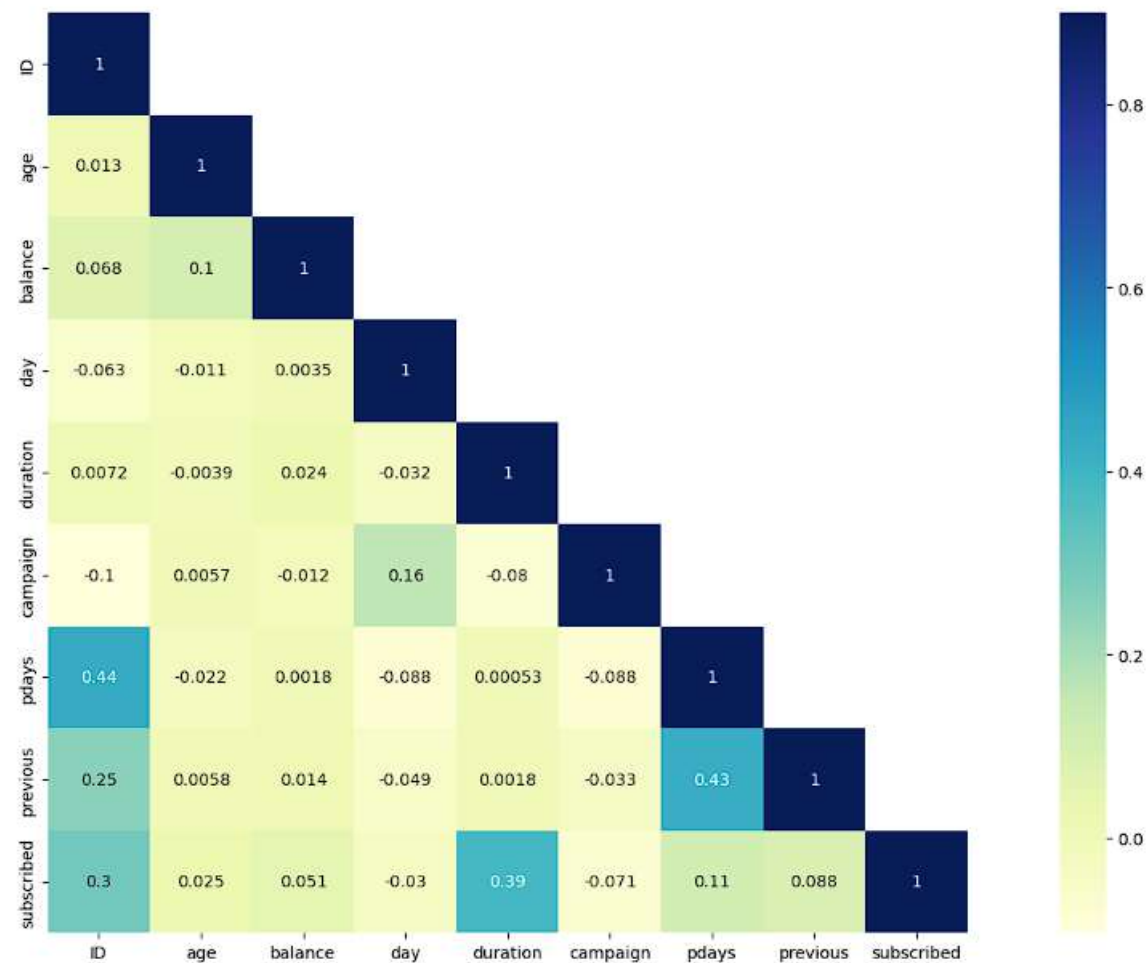
We can infer that clients having no previous default have slightly higher chances of subscribing to a term loan as compared to the clients who have previous default history.

Let's now look at how correlated our numerical variables are. We will see the correlation between each of these variables and the variable which have high negative or positive values are correlated. By this we can get an overview of the variables which might affect our target variable. We will convert our target variable into numeric values first.

```
In [16]: train['subscribed'].replace('no', 0,inplace=True)
train['subscribed'].replace('yes', 1,inplace=True)
```

```
In [17]: corr = train.corr()
mask = np.array(corr)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corr, mask=mask,vmax=.9, square=True,annot=True, cmap="YlGnBu")
```

Out[17]: <Axes: >



We can infer that duration of the call is highly correlated with the target variable. This can be verified as well. As the duration of the call is more, there are higher chances that the client is showing interest in the term deposit and hence there are higher chances that the client will subscribe to term deposit.

Next we will look for any missing values in the dataset.

In [18]: train.isnull().sum()

```
Out[18]: ID      0
         age      0
         job      0
         marital  0
         education 0
         default  0
         balance  0
         housing  0
         loan     0
         contact  0
         day       0
         month     0
         duration  0
         campaign  0
         pdays     0
         previous  0
         poutcome  0
         subscribed 0
         dtype: int64
```

There are no missing values in the train dataset.

Next, we will start to build our predictive model to predict whether a client will subscribe to a term deposit or not.

As the sklearn models takes only numerical input, we will convert the categorical variables into numerical values using dummies. We will remove the ID variables as they are unique values and then apply dummies. We will also remove the target variable and keep it in a separate variable.

Model Building

```
In [19]: target = train['subscribed']
         train = train.drop('subscribed',1)
```

```
In [20]: # applying dummies on the train dataset
         train = pd.get_dummies(train)
```

Now, its time to build our model. We will split the train data into training and validation set so that we will be able to validate the results of our model on the validation set. We will keep 20% data as validation set and rest as the training set.

```
In [21]: from sklearn.model_selection import train_test_split
```

```
In [22]: # splitting into train and validation with 20% data in validation set and 80% data in train set.
         X_train, X_val, y_train, y_val = train_test_split(train, target, test_size = 0.2, random_state=12)
```

Now our data is ready. Its time to build our model and check its performance. Logistic regression is used for classification problems and as it is a classification problem let's first build a Logistic Regression model.

Logistic Regression

```
In [23]: from sklearn.linear_model import LogisticRegression
```

```
In [24]: # defining the logistic regression model
         lreg = LogisticRegression()
```

```
In [25]: # fitting the model on X_train and y_train
         lreg.fit(X_train,y_train)
```

```
In [28]: # calculating the accuracy score
accuracy_score(y_val, prediction)
```

Out[28]: 0.8913112164296998

We got an accuracy score of around 90% on the validation dataset. Logistic regression has a linear decision boundary. What if our data have non linearity? We need a model that can capture this non linearity.

Let's try decision tree algorithm now to check if we get better accuracy with that.

Decision Tree

```
In [29]: from sklearn.tree import DecisionTreeClassifier
```

```
In [30]: # defining the decision tree model with depth of 4, you can tune it further to improve the accuracy score
clf = DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
In [31]: # fitting the decision tree model
clf.fit(X_train,y_train)
```

Out[31]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
In [32]: # making prediction on the validation set
predict = clf.predict(X_val)
```

```
In [33]: # calculating the accuracy score
accuracy_score(y_val, predict)
```

Out[33]: 0.9042654028436019

We got an accuracy of more than 90% on the validation set. You can try to improve the score by tuning hyperparameters of the model. Let's now make the prediction on test dataset. We will make the similar changes in the test set as we have done in the training set before making the predictions.

```
In [34]: test = pd.get_dummies(test)
```

```
In [35]: test_prediction = clf.predict(test)
```

Finally, we will save these predictions into a csv file. You can then open this csv file and copy paste the predictions on the provided excel file to generate score.

```
In [36]: submissio = pd.DataFrame()
```

```
In [37]: # creating a Business_Sourced column and saving the predictions in it
submissio['subscribed'] = test_prediction
```

Since the target variable is yes or no, we will convert 1 and 0 in the predictions to yes and no respectively.

```
In [38]: submissio['subscribed'].replace(0,'no',inplace=True)
submissio['subscribed'].replace(1,'yes',inplace=True)
```

```
In [39]: submissio.to_csv('submissio.csv', header=True, index=False)
```